Automatic Classification of UML Class Diagrams Using Deep Learning Technique: Convolutional Neural Network

Article *in* Applied Sciences · May 2021

DOI: 10.3390/app11094267

Gosala Bethany
Banaras Hindu University
3 PUBLICATIONS **3** CITATIONS
SEE PROFILE

Sripriya Roy Chowdhuri
Banaras Hindu University
3 PUBLICATIONS **3** CITATIONS
SEE PROFILE

Jyoti Singh
Banaras Hindu University
20 PUBLICATIONS **140** CITATIONS
SEE PROFILE

Manjari Gupta
Banaras Hindu University
42 PUBLICATIONS **234** CITATIONS
SEE PROFILE

applied sciences

MDPI

Article

Automatic Classification of UML Class Diagrams Using Deep Learning Technique: Convolutional Neural Network

Bethany Gosala [1], Sripriya Roy Chowdhuri [1], Jyoti Singh [1], Manjari Gupta [1,*] and Alok Mishra [2,3,*]

[1] (Computer Science) DST-Centre for Interdisciplinary Mathematical Sciences, Banaras Hindu University, Varanasi 221005, India; gosala.bethany10@bhu.ac.in (B.G.); sripriya@bhu.ac.in (S.R.C.); jyotisingh4337@gmail.com (J.S.)
[2] Department of Software Engineering, Atilim University, Ankara 06830, Turkey
[3] Faculty of Logistics, Molde University College (Specialized University in Logistics), 6410 Molde, Norway
* Correspondence: manjari@bhu.ac.in (M.G.); alok.mishra@himolde.no (A.M.)

Abstract: Unified Modeling Language (UML) includes various types of diagrams that help to study, analyze, document, design, or develop any software efficiently. Therefore, UML diagrams are of great advantage for researchers, software developers, and academicians. Class diagrams are the most widely used UML diagrams for this purpose. Despite its recognition as a standard modeling language for Object-Oriented software, it is difficult to learn. Although there exist repositories that aids the users with the collection of UML diagrams, there is still much more to explore and develop in this domain. The objective of our research was to develop a tool that can automatically classify the images as UML class diagrams and non-UML class diagrams. Earlier research used Machine Learning techniques for classifying class diagrams. Thus, they are required to identify image features and investigate the impact of these features on the UML class diagrams classification problem. We developed a new approach for automatically classifying class diagrams using the approach of Convolutional Neural Network under the domain of Deep Learning. We have applied the code on Convolutional Neural Networks with and without the Regularization technique. Our tool receives JPEG/PNG/GIF/TIFF images as input and predicts whether it is a UML class diagram image or not. There is no need to tag images of class diagrams as UML class diagrams in our dataset.

Keywords: Unified Modeling Language; Machine Learning (ML); Object-Oriented modeling; Deep Learning (DL); Convolutional Neural Networks (CNN)

check for updates

Citation: Gosala, B.; Chowdhuri, S.R.; Singh, J.; Gupta, M.; Mishra, A. Automatic Classification of UML Class Diagrams Using Deep Learning Technique: Convolutional Neural Network. *Appl. Sci.* **2021**, *11*, 4267. https://doi.org/10.3390/app11094267

Academic Editor: José Carlos Bregieiro Ribeiro

Received: 12 April 2021
Accepted: 4 May 2021
Published: 8 May 2021

1. Introduction

Software development is done in a step-wise manner which can be understood as well-defined stages which we call as Software Development Life Cycle (SDLC). It consists of few stages which properly define all the stages in the development of software. A software cycle deals with various parts and phases from planning to testing and deploying software. All these activities are carried out in different ways, as per the needs. Each way is known as a Software Development Life Cycle Model (SDLC) [1]. This helps the developer to manage and control the phases of development of a particular project. During the process of development, modeling of the software becomes an integral part. Modeling is the process of creating the abstraction of the real-world scenario for which the software is meant to build. For the above-mentioned modeling, Unified Modeling Language (UML) is extensively used. The Unified Modeling Language (UML) is a diagram notation designed to model large systems, particularly large object oriented software systems [2]. UML defines different types of diagrams that represent objects, classes, states, activity and scenario diagrams, etc., with various forms such as rectangle, ellipse, etc., and some defined notations. These diagrams properly depict different aspects of the software system. A class diagram is one such type of UML diagram which helps to express the classes and the relationship between the classes. UML class diagrams (CD) are used to design and illustrate the structure of software. They are a very important tool for engineers to understand the basic structure of a system [3]. UML CDs are becoming ever more prevalent within academia and the industry, where model-driven development is emerging as a common practice, and it is broadly accepted that they have become an integral part [4,5]. The information depicted by class diagrams is the essence of an Object-Oriented Paradigm. Since the Object-Oriented Programming Paradigm is the major base for developing software, class diagrams are most commonly used for modeling the software. Software Engineering proposes different kinds of models such as structural models, behavioral models, etc. Understanding these models and studying them gives the idea of the software system being built, as diagrammatic representations are much more understandable than textual representations. Therefore, it is of major importance to understand the meaning of each kind of model, how they are related and how they evolve [6].

Literature shows it is difficult to learn UML. Siau and Loo [7] discussed five categories of UML learning difficulties. The presence of large public repositories provides great ease to study, analyze, model reuse, or software development as discussed earlier. The availability of such repositories is quite scarce [8]. Few tool vendors (e.g., Sparx Enterprise Architect and Visual Paradigm) provide their private repositories on a commercial basis. However, such private repositories have few drawbacks as they support only support the native file format of the CASE tools used by them and they do not allow any open access to the models generated by some other tools. On the other hand, for the public side, France et al. proposed ReMoDD, a pioneering repository for model-driven development [9,10]. However, those models cannot be easily retrieved because the models are stored as files that have to be downloaded and then opened with a compatible CASE tool otherwise, they are pure PDF files. Users need to search manually for the UML diagrams among thousands of files with limited search capabilities. Successful online model repositories where users can freely access UML models are known to be very few [11]. Therefore, we can conclude although several initiatives have been taken to promote and support the model adoption, we are still far apart from a proper and concrete adoption of such model repositories [12]. Automatic classification of web images was done using a Machine Learning approach [10], which helped in improving the efficacy of common web search engines like Google Images. The approach also proved to be useful to search the repositories when metadata is not available. However, with the advancement in the domain of Deep Learning, the need of the hour is to explore more and more techniques in this domain for the classification of UML class diagrams and ultimately improvise software development.

1.1. Problem Statement

Few researchers have done empirical studies and developed tools for automatically classifying the UML class diagrams. They got different results. Since those studies used Machine Learning techniques, their results depend on the image features they considered and the Machine Learning approaches they used. Deep learning algorithm advances the data through different levels and subsequently, features are extracted. Initial layers extract low-level features, and succeeding layers combine features to form a complete representation [13]. In this paper, we try to explore Deep Learning techniques, which is a broader aspect of Machine Learning based on Artificial Neural Networks, for solving the UML CDs classification problem. We applied Convolutional Neural Networks (CNN) which are mainly used for images. CNN uses relevant filters to capture the spatial and temporal dependencies in an image [14].

1.2. Motivation

Many factors have motivated us to perform this study. Major factors are discussed below:

1. Researchers want to study the effects of UML modeling on different software development phases, particularly on the maintenance phase. As many empirical studies in Software Engineering are done on open source projects, the automated extraction and

analysis of large sets of project data become possible. For researching the effects of UML modeling in open source projects, we need a way to automatically determine the type of UML use in a project [15].

2. Nowadays, there are many UML CASE tools, which give many features like creating and modifying UML models, exporting models into XMI files, export models into images and automatically generating the user code backbone. UML CASE tools typically can work with different file types for storing UML models. Unfortunately, these file types cannot be exchanged with another one. Until now, CASE tools do not support extracting UML models from images. Thus, there is a need to automatically extract UML Class models from images without redrawing the model using a CASE tool [7].
3. As discussed above, it is not easy to learn UML, and because of that, creating a repository of different diagrams of UML will help novice developers to learn from the experience of expert developers who created UML diagrams that were used to develop good quality software and kept in the repository. Many different kinds of UML images on the web make the possibility to retrieve their modeling information for reuse purposes exciting.
4. Different UML diagrams need to be analyzed because of several reasons. Identifying syntactic inconsistencies is one of the reasons. To analyze the supposed class diagrams, it would be required to automatically classify class diagrams from the repository of different UML diagrams.

1.3. Research Questions

Here, in this section, we will present our Research Questions that we are going to address in our research paper.

RQ1: Do Deep Learning techniques, particularly Convolutional Neural Networks (CNN), improve the accuracy of automatic classification of UML class diagrams and non-UML class diagrams?

RQ2: How do various parameters affect the accuracy of the algorithm?

1.4. Research Objectives and Contributions

The contributions of our paper are as follows:

1. This is one of the first works on UML class diagrams classification where the use of the Deep Learning approach is explored. Since we are directly working on images, Convolutional Neural Network (CNN) approach is the best approach to apply.
2. We are providing the accuracy of our algorithm for both the train set and test set. Previous approaches by B. Karasneh et al. [16] and Ho-Quang et al. [3] mentioned the word "accuracy" only. They have not mentioned whether it is the accuracy of the train set or test set. Thus, we are assuming they are getting the same accuracy on both datasets. Our accuracy on the train set is greater than the accuracy of other approaches. Accuracy on the test set is slightly less than a few of other's accuracy. The reason behind this is that the dataset is small for Deep Learning technique as these techniques give a good result on large datasets.

The remaining paper has "six" more sections to follow. In Section 2, we discuss the background study of the paper. In Section 3, we explain the related work behind the paper. In Section 4, we explore the solution approach and research framework. In Section 5, we give the details about the experiment and results obtained. In Section 6, we have the discussion, and in Section 7 we have the conclusion and future scope.

2. Background

UML defines different types of diagrams in the form of graphs that represent objects, classes, states, etc., with various forms such as rectangle, ellipse, etc., and some defined notations. They also contain some texts, labeled arcs, and arrowheads. These diagrams properly depict different aspects of the software system. UML diagrams can also model the different software engineering models such as structural, behavioral, or design models. These models help the researchers or developers to understand the system properly.

Automatic classification of UML sequence diagrams [17] was done to enhance the UML repository so that researchers and academicians can share and study UML artifacts for better development of software. Analyzing and deciphering UML models and sharing modeling artifacts is a rising need. Simple drawing tools or fully developed computer-assisted software engineering (CASE) tools can be used to produce UML diagrams, there are several means.

UML diagrams or models can be stored in repositories that can be used by the researchers and academicians and developers etc. for their sole purpose of studying, analyzing, or industrial developments. Certain repositories serve the purpose but still, they are not up to the mark because some of them mostly support the basic file format of the CASE tool used, and also enriching such repositories with properly classified UML class diagrams and non-UML diagrams become difficult. This is because there are enormous images present over the web, and the correct classification of those images with maximum accuracy becomes quite a challenging task. In addition, the limited search capabilities of the repositories intend the users to manually search UML diagrams from thousands of files.

A huge and enormous number of digital documents and images are available in databases over the internet. Due to such large volumes of digital information, human capability to interpret digital images and documents becomes very difficult and thus an automated system is required to capture the details more effectively and efficiently. Studying UML models and sharing modeling artifacts [18] is an emerging need in recent years. To make this possible, UML diagrams are needed to be collected in a repository of some kind. Some general model repositories [9] have been constructed. These repositories need to be augmented with UML diagrams which can be used by the researchers and academicians and developers etc. for their sole purpose of studying, analyzing, developing, or reusing software. Although enriching such repositories is not that difficult since there are enormous images present over the web, however classifying those images as UML diagrams and collecting the relevant diagrams in the repositories is quite a difficult task. This is because nowadays we are having vast and enormous digital resources. Therefore, automated classification of images as UML class diagrams and collecting them in the repositories has become a topic of researchers' interest.

Researchers have contributed to classifying the web images as UML class diagrams and non-UML diagrams by using different techniques including the Machine Learning paradigm. In 1959, Arthur Samuel, a pioneer in the field of Machine Learning (ML) defined it as the "field of study that gives computers the ability to learn without being explicitly programmed". All the techniques which have been applied previously are Machine Learning techniques. Nowadays, data are becoming more versatile and in various forms so it is good to apply Deep Learning techniques to get good accuracy on multi-variant data, hence we are trying to apply the Deep Learning technique.

2.1. Unified Modeling Language (UML)

Unified Modeling Language (UML) is a modeling language developed to simplify the software design process. UML is a generalized and standard structure including a flexible graphical notification to create visual models/diagrams of basically object-oriented software systems.

Unified Modeling Language was designed and developed by Grady Booch, Ivar Jacobson, and James Rumbaugh at Rational Software in 1994–1995, with further development led by them through 1996. The UML 1.0 specification draft was proposed to the Object Management Group (OMG) in January 1997.

UML defines different types of diagrams which represent objects, classes, states, etc., with various forms such as rectangle, ellipse, etc., and some defined notations. These diagrams properly help to visualize, analyze, and understand different aspects of the software system based on the Object-Oriented Paradigm. UML includes several types of diagrams such as use case diagrams, sequence diagrams, class diagrams, component diagrams, deployment diagrams, activity diagrams, object diagrams, etc. Class diagrams are the most commonly used UML diagrams. They help to express the classes and the relationship between the classes and the objects. UML class diagrams (CD) are used to design and illustrate the structure of software. They are a very important tool for engineers to understand the basic structure of a system [3].

2.2. Deep Learning

Deep Learning is a form of Machine Learning that enables computers to learn from experience and understand the world in terms of a hierarchy of concepts. As a result that the computer gathers knowledge from experience, there is no need for a human–computer operator formally to specify all of the knowledge needed by the computer [19].

2.3. Neural Networks

An Artificial Neural Network (ANN), usually called Neural Network (NN), is a mathematical model or computational model that is inspired by the structure and/or functional aspects of biological neural networks. A neural network consists of an interconnected group of artificial neurons, and it processes information using a connectionist approach to computation [20]. A basic Neural Network has an input layer followed by one or more hidden layers and an output layer as shown in Figure 1.

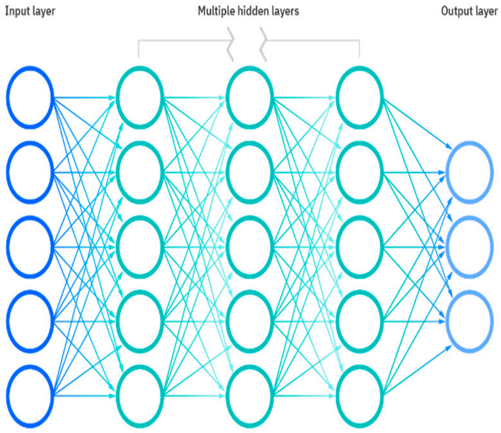

Figure 1. Neural Network.

2.4. Activation function

Derivatives are primary essentials for the optimization of any Neural Network. Activation functions are the functions that allow non-linearity in the fundamentally linear models, in other words, sequence of linear operations. The activation function which we used in our model is the most commonly used one in the Neural Networks, which is Rectified Linear Unit (ReLU), shown in Figure 2. It is defined as R(z) = max {0, z}. The ReLU activation function is not differentiable at z = 0. ReLU is differentiable at all the points except 0, the left derivative at z = 0 is 0 and the right derivative is 1 [21].

2.5. Google-Colab

"Colab" is a short form of collaboratory and is a Google Research product. Colab allows anyone to write and execute code through the web browser. It suits well for Machine Learning and data analytics. Colab is a hosted Jupyter notebook service that does not require any setup to run. It provides free access to computing resources including GPUs.

2.6. Tensor Flow

TensorFlow is an open-source software library developed by Google Brain Team for numerical computations, particularly well suited and fine-tuned for large-scale Machine Learning. In TensorFlow, it is possible to break up the graph into several chunks and run them parallel across multiple CPUs or GPUs [22].

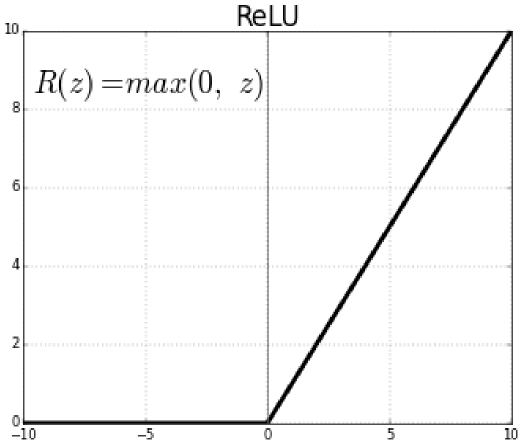

Figure 2. ReLU activation function.

3. Related Work

An online repository of CDs was proposed by [3]. They proposed 23 image features and investigated the use of these features to classify UML CD images. Six classification algorithms were used on a set of 1300 images. They found that 19 out of 23 introduced features can be considered as influential predictors for classifying UML CD images. Enriching such a repository with images was quite easier because of the availability of a large number of UML CDs as images on the Internet. However, the problem with the above-mentioned approach was the proper classification of the images according to the relevance.

Bislimovska et al. [23] proposed a content-based and keyword-based retrieval of models from repositories. This proves to be good testimony of the importance of retrieval techniques and repositories in model-driven engineering. However, this research is focused only on textual elements of models, i.e., the models are needed to be stored in different textual formats, such as XML (Extensible Markup Language) or its derivations (such as XMI (XML Metadata Interchange). These do not consider the model interpretation stored as images.

A recent and substantial study of UML models stored in different formats in GitHub, combining a lot of manual work and automatic processing recognized 93,596 UML models from 24,717 different repositories, of which 61.8% (57,822) are images, the rest being files with extensions, XML or UML [24]. These data confirm that the existence of interesting and potentially useful information in repositories are not up to the mark which makes it difficult to access and reuse; this also gives the idea that a huge proportion of such models are stored as images [10].

The interest in automatic recognition of UML diagrams in repositories dates back to the times of appearance of the UML, with emphasis on sketch recognition in the software design cycle, to make available the information they contained to CASE tools [1]. The MDD/MDE (Model-Driven Development/Model-Driven Engineering) community supports and motivates the effective reuse of modeling artifacts and managing vast repositories, and this has increased the need for the identification of tool-made diagrams [12]. According to our knowledge, there is much more to explore in this area by the researchers. The research done in this field is quite scarce with only a few using Machine Learning techniques. The majority of the approaches use image processing and pattern recognition methods modified according to specific domains, such as architectural diagrams [25] or finite-state automata diagrams [26].

In the field of UML diagrams, Karasneh and Chaudron have research published about the same problem [27], although they have not used any automatic learning approach, but used a fixed set of classification criteria. Their Img2UML tool [16] uses the Aforge.NET framework which is an open-source C# framework designed for researchers and developers in the fields of Artificial Intelligence, Neural Networks, Machine Learning, robotics, computer vision, image processing, genetic algorithms, fuzzy logic, etc. The UML class models are extracted from pixmap images and exported into XMI files by the tool and those files can be read by a commercial CASE tool (StarUML).

A private repository is created with the collected images from the internet along with their XMI extraction [28]. Although this work has covered a broader aspect of the problem in this field still the tool does not solve all the problems which have been identified by them (specifical identification of relationships, OCR recognition, and lack of XMI standardization). They used a few images to authenticate their tool (10 and 200 images in [27] and [28], respectively) and giving an accuracy of 89%.

In the work of Ho-Quang et al. [3], a Machine Learning approach is introduced and they used 23 image features to train an automatic classifier of class diagrams (CD) using the Support Vector Machine (SVM) algorithm, founded on computer vision techniques enforced with aid of the open-source image-processing libraries Magick++ and OpenCV. They have used a training set of 1300 images downloaded from different online sources and equally divided them into CD and non-CD, with 10 s of average time to process each image and extract the graphical features used by the classification algorithm, reaching a 91.2% accuracy of correct classifications.

The former works were continued by Ho-Quang et al. [3], who further examined image features that can be efficaciously used to classify images as UML class diagrams and to train an automatic classifier. They successfully reduced the image features count from 23 to 19 and the processing time from 10 s to 5.84 s, by applying six different types of learning algorithms on the same training set of 1300 images. Each one of the algorithms resulted in the range of 90.7% to 93.2% accuracy.

However, the authors considered specificity followed by sensitivity as the most crucial criterion of effectiveness, and hence they consider Random Forest and Logistic Regression as their best nominee. Finally, Hebig et al. [29] used the same automatic tool developed by Ho-Quang et al. [3] with a new set of 19,506 images collected from GitHub and manually classified as CD and non-CD. They did not train the tool again and used it as-is to automatically classify the new set of images and compare the accuracy obtained with previous results. Among the six learning algorithms formulated by Ho-Quang et al. [3], they selected Random Forest for its best performance.

Ott et al. [30] worked on low shot learning in mining software repositories. They proposed an approach of simpler deep architectures in combination with low shot learning to reduce the learning parameters for classifying UML Class and Sequence diagrams. Moreno et al. [10] developed a method to automatically and efficiently classify web images as Unified Modeling Language (UML) static diagrams and to produce a computer tool that whether the image corresponds to a diagram. The tool selects graphical characteristics from every image (color histogram, grayscale histogram, and elementary geometric forms) and uses a combination of rules obtained using Machine Learning techniques to classify the images.

Best et al. [31] explored the applicability of transfer learning utilizing models that are pre-trained on non software engineering data to the problem of classifying UML diagrams. Their results show training reacts positively to transfer learning as related to sample size, even though the pre-trained model was not exposed to training instances from the software domain. They contrast the transferred network with other networks to show its advantage on different sized training sets, which indicates that transfer learning is equally effective to custom deep architectures in respect to classification accuracy when large amounts of training data is not available.

Mangaroliya et al. [32] proposed a classification model that classifies a UML diagram into forward engineered class diagrams or reverse engineered class diagrams. They used supervised Machine Learning technique. Their approach analyzed the features and used the best features in classifying class diagrams. They used different Machine Learning models for this process and found Random Forest algorithm to be the best out of all with 96% accuracy. Performance testing was done on 999 class diagrams.

Nanthaamornphong et al. [33] developed and evaluated a software tool (ForUML) to extract Unified Modeling Language (UML) class diagrams from Fortran code. ForUML uses a reverse engineering approach to transform Fortran source code into UML models. ForUML integrates the capabilities of ArgoUML to visually display the class diagram and displays the results of the analysis (the UML class diagram). The accuracy of ForUML was evaluated using five CSE software packages that use object-oriented features from the Fortran 95, 2003, and 2008 compiler standards.

To the best of our knowledge, there is not much work done in the domain of applying Deep Learning algorithm is applied to classify CDs from the set of different types of diagrams and hence there is a wide scope for the researchers to work in this domain. In this paper, we used the CNN approach and experiment with different parameters used in the approach to find an effect of these parameters on the results.

4. Solution Approach and Research Framework

In this section, we will be explaining the dataset we have used in our study, the experiments we have conducted, and the various types of metrics we have used in our experiment.

4.1. Data-Description

For our experiment purpose, we are using the dataset from a Conference paper called Automatically Classifying UML Class Diagrams from Images using "Deep Learning" Anonymous. (2021) [34]. The dataset consists of a total of 3298 images out of which 1649 images are UML class diagrams and 1649 images are non-UML class diagrams. We modified the dataset by deleting few duplicate images and thus the updated dataset consists of 3282 images among which 1635 are UML class diagrams and 1647 are non-UML class diagrams. Non-UML class diagrams consist of images that belong to various classes such as, bar charts, pie-charts, flow charts, histograms, map images, images of animals, birds, and human images.

4.2. Algorithm and Evaluation Measure

We applied a Deep Learning technique called Convolutional Neural Networks (CNN). We have run the code on Convolutional Neural Networks with and without regularization techniques. The model is shown in Figure 3.

1. For CNN without regularization, we used four convolutional layers, four pooling layers. Data augmentation is used to avoid overfitting. Two activation functions, ReLU and Sigmoid, are used. ReLU is used in the inner layers of CNN and Sigmoid is used in the output layer of CNN. Adaptive momentum estimation (Adam) is used for optimization and Max-pooling is used in the pooling layer.
2. For CNN with regularization, we used five CNN layers and five pooling layers. ReLU and Sigmoid activation functions are used. Adam Optimizer, Max-pooling, and L2 regularization with different weight functions are used for the evaluation. We calculate the accuracy every time we run the code and the best accuracy on the test set is taken into the account.

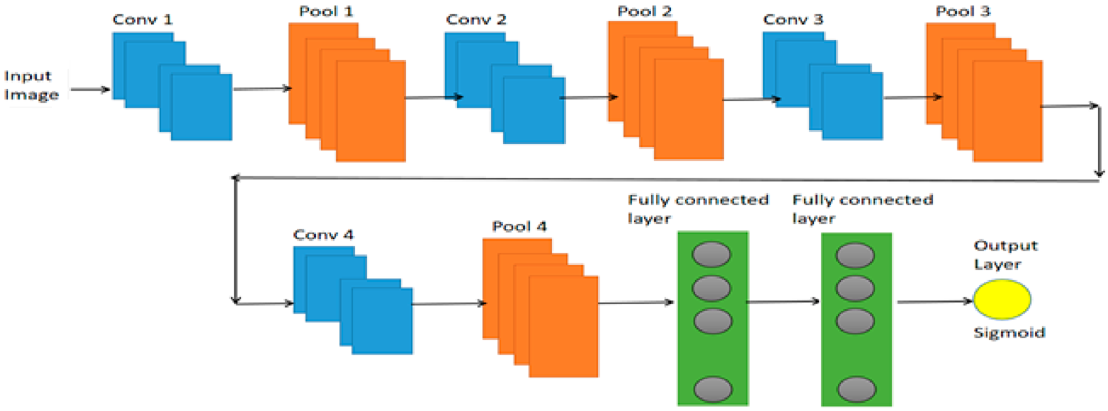

Figure 3. Convolutional Neural Network.

4.3. *Experiment Settings*

This is a sub-section where the experiment settings are explained. The dataset is divided into two parts called training set and test set, each consisting of 90% and 10% of the dataset, respectively. We have used Googles' cloud service "colab" to run the code, which provides tesla K80 GPU. We have run the code several times with different parameters and used the optimizer: adaptive momentum estimation (Adam). To avoid over-fitting, we have used data augmentation.

4.4. *Data Augmentation*

Data augmentation will also help us to generate more data. We have normalized the image pixel-wise by dividing each image by 255. We have applied various arguments like rotation, width-shift, height-shift, shear, zoom, etc., for the augmentation. We used MaxPool in the pooling layer. The best results are discussed in the experimental results section.

4.5. *Framework*

The construction of the model and the framework of the model is shown in Figure 4.

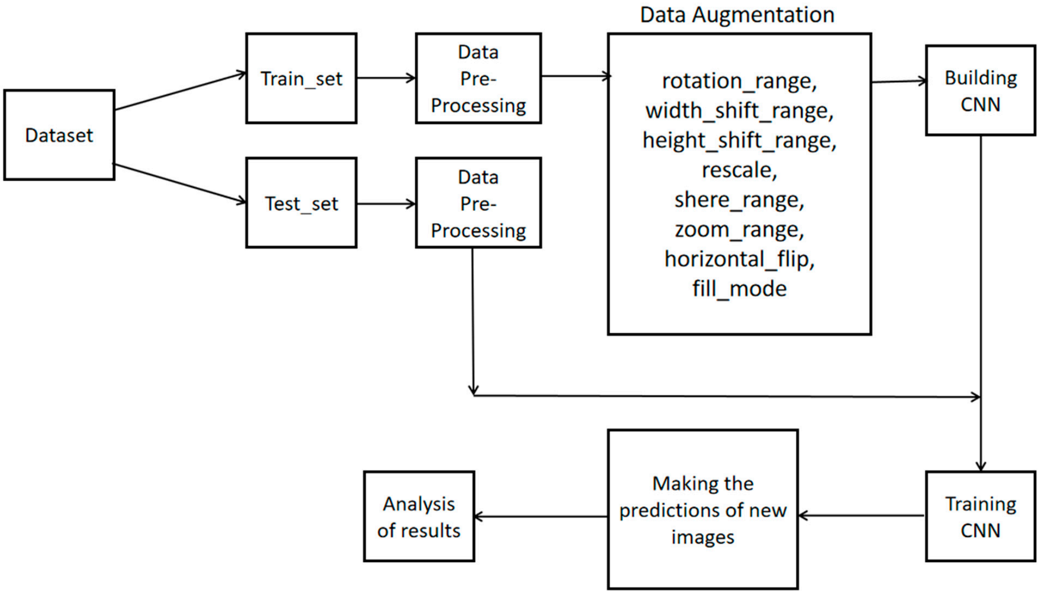

Figure 4. The overall framework of building the model.

5. Experiments and Results obtained

After running the code on CNN with regularization and without regularization, our best results are as follows:

i. For CNN without regularization, we got an accuracy of 94.51% on the train set and 85.41% on the test set.
ii. For CNN with regularization, we achieved 96.06% and 86.63% accuracy on train and test sets, respectively.

5.1. Analysis of Results

RQ1: For our first RQ, we can answer in this way: Deep Learning methods allow us to have a greater number of features, filters, and parameters that can yield the best and most accurate results. By using Convolutional Neural Networks, we can have a variety of parameters that helps us to improve the accuracy, increasing the number of hidden layers, using different types of optimizers, and varying the learning rate all the factors contributes to the accuracy.

RQ2: The parameters have a great impact on the accuracy. We conducted eight experiments. We have run the two algorithms discussed above four times each.

5.1.1. CNN without Regularization

Experiment 1: The size of the image is 150 × 150 pixels. The CNN consists of 11-hidden layers. We used Adam Optimizer. The size of the Kernel is 3 × 3. The accuracy we got was 85.99 on the train set and 79.49 on the test set.

Experiment 2: The size of the image is 128 × 128 pixels. The CNN consists of 13-hidden layers. We used Adam Optimizer. The size of the Kernel is 2 × 2. The accuracy we got was 85.61 on the train set and 82.91 on the test set.

Experiment 3: The size of the image is 150 × 150 pixels. The CNN consists of 13-hidden layers. We used Adam Optimizer. The size of the Kernel is 2 × 2. The accuracy we got was 94.99 on the train set and 85.11 on the test set.

Experiment 4: The size of the image is 128 × 128 pixels. The CNN consists of 11-hidden layers. We used Adam Optimizer. The size of the Kernel is 2 × 2. The accuracy we got was 94.51 on the train set and 85.41 on the test set.

Effect of various parameters of CNN without regularization is shown in Table 1.

Table 1. Effect of various parameters of CNN without regularization.

	Image Size	**Layers**	**Optimizer**	**Kernel**	**Accuracy**
Experiment 1	150 × 150	11	Adam	3 × 3	Train: 85.99 Test: 79.49
Experiment 2	28 × 128	13	Adam	2 × 2	Train: 85.61 Test: 82.91
Experiment 3	150 × 150	13	Adam	2 × 2	Train: 94.99 Test: 85.11
Experiment 4	28 × 128	11	Adam	2 × 2	Train: 94.51 Test: 85.41

5.1.2. CNN with Regularization

Experiment 5: The size of the image is 128 × 128 pixels. The CNN consists of 11-hidden layers. We used Adam Optimizer. The size of the Kernel is 2 × 2, regularization L2 with a learning rate (1×10^{-4}). The accuracies we got were 93.11 on the train set and 84.80 on the test set.

Experiment 6: The size of the image is 150 × 150 pixels. The CNN consists of 15-hidden layers. We used Adam Optimizer. The size of the Kernel is 2 × 2, regularization L2 with a learning rate (1×10^{-6}). The accuracies we got were 93.28 on the train set and 85.41 on the test set.

Experiment 7: The size of the image is 128 × 128 pixels. The CNN consists of 13-hidden layers. We used Adam Optimizer. The size of the Kernel is 3 × 3, regularization L2 with a learning rate (1×10^{-7}). The accuracies we got were 93.25 on the train set and 86.32 on the test set.

Experiment 8: The size of the image is 150 × 150 pixels. The CNN consists of 11-hidden layers. We used Adam Optimizer. The size of the Kernel is 3 × 3, regularization L2 with a learning rate (1×10^{-6}). The accuracies we got were 96.06 on the train set and 86.63 on the test set.

Effect of various parameters of CNN with regularization is shown in Table 2.

Table 2. Effect of various parameters of CNN with regularization.

	Image Size	Layers	Optimizer	Kernel	Regularization	Accuracy
Experiment 5	128 × 128	11	Adam	3 × 3	L2(1×10^{-4})	Train: 93.11 Test: 84.80
Experiment 6	150 × 150	15	Adam	2 × 2	L2(1×10^{-6})	Train: 93.28 Test: 85.41
Experiment 7	128 × 128	13	Adam	2 × 2	L2(1×10^{-7})	Train: 93.25 Test: 86.32
Experiment 8	128 × 128	11	Adam	2 × 2	L2(1×10^{-6})	Train: 96.06 Test: 86.63

6. Discussion

There are few fields that we have explored in this paper, which are not much explored in earlier studies, like the use of Deep Learning in UML CDs, Neural Networks (NN), etc., We have successfully implemented the models which are proposed by us. Though the model has few layers and fewer parameters, the results which we got are quite encouraging. Our model worked well on the train set and got the best results when it is compared to all the previous models, even the results on the test set are also good but not exactly up to the mark.

6.1. Threats to Validity

In this section, we discuss the possible threats that might have affected our approach and how we mitigated them. Validity is the extent to which the results measure what they are supposed to measure.

6.1.1. Threats to Internal Validity

The fundamental threat to internal validity is the dataset itself. The dataset used in our approach contains 3282 images only which is very small to apply Deep Learning techniques. The distributions of the train set and test set are also not perfect because, in general, the Deep Learning dataset must be very big—at least in 10,000 s—to acquire desired results.

6.1.2. Threats to Conclusion Validity

This threat centers on evaluating the performance of the prediction models, i.e., it affects the ability to draw correct conclusions about relations between treatment and outcome. In this approach, we have used only one Deep Learning technique called Convolutional Neural Networks. Better results can be found if we apply more Deep Learning techniques.

6.1.3. Threats to External Validity

External validity limits the ability to generalize the results beyond the experiment setting. There is not much variety in the non-UML class diagrams, many of the images are very similar to other images. Hence, the generalization could not be made properly. The images which are fed to the algorithm for the training and testing have a huge difference among the UML class diagrams and non-UML class diagrams. If you want good accuracy on a test set and to get the difference between train set accuracy and test set accuracy as low as possible there should be very little difference between the positive UML class diagrams and negative UML class diagrams. This will help the model to generalize rather than memorize.

Karasneh et al. [16] have used a dataset of 200 images and got an accuracy of 89%, Ho-Quang et al. [3] got an accuracy of 91.2% on the dataset of 1300 images and they used the SVM algorithm. Ho-Quang et al. [3] got up to 93.2% accuracy on 1300 images in their dataset, Valentin Mareno et al. [10] developed a model called image-to-model (i2m) using a dataset of 18,899 images and got an accuracy of 94.4%. Ott Jordan et al. [30] have used the technique of low-short learning which is very useful when the dataset is very small and combined with the Convolutional Neural Network technique on the dataset compressing of 11,319 images to classify UML class diagrams from sequence diagram, and they claimed accuracy of more than 92%. In our work, we used the dataset of 3282 images and got an accuracy of 94.51 on the train set and 85.41 on the test set with the first model, which was CNN without regularization, an accuracy of 96.06% on the train set and 86.63 on the test set for second model, which was CNN with regularization.

Table 3, given below, shows the previous studies and results and the approach they have followed, and also shows the comparison of their accuracies with the accuracy of our research. The reason for differences in accuracies are mainly because of size of the dataset, images used in the datasets, and algorithm(s) used by different approaches.

Table 3. Accuracy comparison with our paper.

S.No.	Authors	Number of Images	Algorithm	Accuracy
1.	Karasneh et al. [16] (not a Machine Learning approach)	200	N/A	89.0%
2.	Ho-Quang et al. [3]	1300	SVM	91.2%
3.	Ho-Quang et al. [3]	1300	Random Forest	90.7–93.2%
4.	Hebig et al. [29] (Not a Machine Learning approach)	19,506	N/A	N/A
5.	Valentin Mareno et al. [10]	18,899	image-to-model (i2m)	94.4%
6.	Ott j et al. [30]	11,319	Low-short learning with Convolutional Neural Networks	More than 92%
7.	Our research	3282	CNN without regularization	Train: 94.51 Test: 85.41
			CNN with regularization	Train: 96.06 Test: 86.63

The results that we obtained can still be improved but due to the size of the dataset we just have to contend with the results. The findings that we obtained will certainly encourage the other researchers who want to work in this area and who want to use Deep Learning techniques.

7. Conclusions and Future Research

In this paper, we have presented an automatic classification method that classifies UML class diagrams. In our work, we used CNN for classifying UML class diagrams. Previous approaches (Karasneh et al. [16], Ho-Quang et al. [3]) have used Machine Learning techniques and mentioned the word "accuracy" only in their results. They have not mentioned whether it is the accuracy of the train set or test set. Thus, we are assuming they are getting the same accuracy on both datasets. We compared our results with other existing techniques and found that CNN results in better accuracy on training sets than other Machine Learning techniques for this problem. Accuracy for test sets is slightly less than some other approaches. The accuracy which we got on test sets was 85.41 and 86.63 for CNN without and with regularization, which is less as compared with other Machine Learning methods. This is because of a very small dataset. Generally, for Deep Neural Networks to result in good accuracy, the dataset must be large.

There is a need to generate a big dataset so that a larger variety of Deep Learning techniques can be applied to reduce the error rate and to get human-level accuracy. However, in the absence of a large dataset, a transfer learning approach can be considered which allows training deep architectures even if limited data are available [31]. Furthermore, the application of image processing tools, along with the Deep Learning algorithm will help to achieve optimal results. In another direction, there is scope to extend this model using Natural Language Processing (NLP) for the semantic understanding of diagrams. Text-recognition features will also play a vital role.

Here, in this paper, we concentrated on classifying only one of the types of UML diagrams which is the UML class diagram from all other images (UML or non-UML diagrams). The scope can be extended to classify various UML diagrams such as structural UML diagrams (composite structure diagrams, deployment diagrams, package diagrams, profile diagrams, class diagrams, object diagrams, component diagrams), and behavioral UML diagrams (static machine diagrams, communication diagrams, use case diagrams, activity diagrams, sequence diagrams, timing diagrams, interaction overview diagrams).

Author Contributions: Conceptualization, M.G. and B.G.; data curation, B.G.; formal analysis, B.G., S.R.C. and J.S.; investigation, B.G.; methodology, M.G. and A.M.; supervision, M.G. and A.M.; validation, B.G., S.R.C. and J.S.; visualization, B.G., S.R.C. and J.S.; writing—original draft, B.G., S.R.C. and J.S.; writing—review and editing, M.G. and A.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The dataset is available at the following link: https://zenodo.org/record/4252890#.X8JFjM0zbIU.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Code Better. Com-Stuff You Need to Code Better! Available online: http://codebetter.com/raymondlewallen/2005/07/13/software-development-life-cycle-models/ (accessed on 13 June 2005).
2. Lank, E.; Thorley, J.; Chen, S.; Blostein, D. On-line recognition of UML diagrams. In Proceedings of the Sixth International Conference on Document Analysis and Recognition, Seattle, WA, USA, 10–13 September 2001; pp. 356–360.
3. Ho-Quang, T.; Chaudron, M.R.; Samuelsson, I.; Hjaltason, J.; Karasneh, B.; Osman, H. Automatic classification of UML class diagrams from images. In Proceedings of the 21st Asia-Pacific Software Engineering Conference, Jeju, Korea, 1–4 December 2014; Volume 1, pp. 399–406.
4. Thramboulidis, K.C. Using UML in control and automation: A model driven approach. In Proceedings of the 2nd IEEE International Conference on Industrial Informatics, Berlin, Germany, 24–26 June 2004; pp. 587–593.
5. Secchi, C.; Fantuzzi, C.; Bonfe, M. On the use of uml for modeling physical systems. In Proceedings of the 2005 IEEE International Conference on Robotics and Automation, Barcelona, Spain, 18–22 April 2005; pp. 3990–3995.
6. Énova, G.; Valiente, M.C.; Marrero, M. On the difference between analysis and design, and why it is relevant for the interpretation of models in Model Driven Engineering. *J. Object Technol.* **2009**, *8*, 107–127.
7. Siau, K.; Loo, P.P. Identifying difficulties in learning UML. *Inf. Syst. Manag.* **2006**, *23*, 43–51. [CrossRef]
8. Karasneh, B.H.A. An Online Corpus of UML Design Models: Construction and Empirical Studies. Ph.D. Thesis, Leiden University, Leiden, The Netherlands, 7 July 2016.
9. France, R.; Bieman, J.; Cheng, B.H. Repository for model driven development (ReMoDD). In *Proceedings of the International Conference on Model Driven Engineering Languages and Systems, Genoa, Italy, 1–6 October 2006*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 311–317.
10. Moreno, V.; Génova, G.; Alejandres, M.; Fraga, A. Automatic Classification of Web Images as UML Static Diagrams Using Machine Learning Techniques. *Appl. Sci.* **2020**, *10*, 2406. [CrossRef]
11. Whittle, J.; Hutchinson, J.; Rouncefield, M.; Burden, H.; Heldal, R. A taxonomy of tool-related issues affecting the adoption of model-driven engineering. *Softw. Syst. Model.* **2017**, *16*, 313–331. [CrossRef]
12. Basciani, F.; Di Rocco, J.; Di Ruscio, D.; Iovino, L.; Pierantonio, A. *Model Repositories: Will They Become Reality?* CloudMDE@MoDELS: Ottawa, ON, Canada, 2015; pp. 37–42.

13. Mathew, A.; Amudha, P.; Sivakumari, S. Deep Learning Techniques: An Overview. In *Proceedings of the International Conference on Advanced Machine Learning Technologies and Applications, Jaipur, India, 13–15 February 2020*; Springer: Singapore, 2020; pp. 599–608.
14. Yamashita, R.; Nishio, M.; Do, R.K.; Togashi, K. Convolutional neural networks: An overview and application in radiology. *Insights Imaging* **2018**, *9*, 611–629. [CrossRef] [PubMed]
15. Osman, M.H.; Ho-Quang, T.; Chaudron, M. An automated approach for classifying reverse-engineered and forward-engineered UML class diagrams. In Proceedings of the 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Prague, Czech Republic, 29–31 August 2018; pp. 396–399.
16. Karasneh, B.; Chaudron, M.R. Img2uml: A system for extracting uml models from images. In Proceedings of the 39th Euromicro Conference on Software Engineering and Advanced Applications, Santander, Spain, 4–6 September 2013; pp. 134–137.
17. Rashid, S. Automatic Classification of uml Sequence Diagrams from Images. Bachelor's Thesis, University of Gothenberg, Gothenberg, Sweden, 12 November 2019.
18. Buse, R.P.; Zimmermann, T. Information needs for software development analytics. In Proceedings of the 34th International Conference on Software Engineering (ICSE), Zurich, Switzerland, 2–9 June 2012; pp. 987–996.
19. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016; pp. 168–224.
20. D'Addona, D.M. Neural Network. In *The International Academy for Production Engineering*; Laperrière, L., Reinhart, G., CIRP Encyclopedia of Production Engineering, Eds.; Springer: Berlin/Heidelberg, Germany, 2014. [CrossRef]
21. ReLU: Not a Differentiable Function: Why Used in Gradient Based Optimization? And Other Generalizations of ReLU. Available online: https://medium.com/@kanchansarkar/relu-not-a-differentiable-function-why-used-in-gradient-based-optimization-7fef3a4cecec (accessed on 15 November 2019).
22. Géron, A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 2nd ed.; O'reilly Media: Newton, MA, USA, 2019.
23. Bislimovska, B.; Bozzon, A.; Brambilla, M.; Fraternali, P. Textual and content-based search in repositories of web application models. *ACM Trans. Web (TWEB)* **2014**, *8*, 1–47. [CrossRef]
24. Robles, G.; Ho-Quang, T.; Hebig, R.; Chaudron, M.R.V.; Fernández, M.A. An extensive dataset of UML models in GitHub. In Proceedings of the 14th International Conference on Mining Software Repositories, Beunos Aires, Argentina, 20–21 May 2017; pp. 519–522.
25. Maggiori, E.; Gervasoni, L.; Antúnez, M.; Rago, A.; Pace, J.A. Towards Recovering Architectural Information from Images of Architectural Diagrams. In Proceedings of the 15th Argentine Symposium on Software Engineering (ASSE), Buenos Aires, Argentina, 8 September 2014.
26. Babalola, O. Automatic Recognition and Interpretation of Finite-State Automata Diagrams. Master's Thesis, Stellenbosch University, Stellenbosch, South Africa, 14 December 2015.
27. Karasneh, B.; Chaudron, M.R. Extracting UML models from images. In Proceedings of the 5th International Conference on Computer Science and Information Technology, Amman, Jordan, 27–28 March 2013; pp. 169–178.
28. Karasneh, B.; Chaudron, M.R.V. Online Img2UML repository: An online repository for UML models. In Proceedings of the International Workshop on Experiences and Empirical Studies in Software, Miami, FL, USA, 1 October 2013; pp. 61–66.
29. Hebig, R.; Ho-Quang, T.; Chaudron, M.R.V.; Robles, G.; Fernández, M.A. The quest for open source projects that use UML: Mining GitHub. In Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, Saint-Malo, France, 2–7 October 2016; pp. 173–183.
30. Ott, J.; Atchison, A.; Linstead, E.J. Exploring the applicability of low-shot learning in mining software repositories. *J. Big Data* **2019**, *6*, 1–10. [CrossRef]
31. Best, N.; Ott, J.; Linstead, E.J. Exploring the efficacy of transfer learning in mining image-based software artifacts. *J. Big Data* **2020**, *7*, 1–10. [CrossRef]
32. Mangaroliya, K.; Patel, H. Classification of Reverse-Engineered Class Diagram and Forward-Engineered Class Diagram using Machine Learning. *arXiv* **2020**, arXiv:2011.07313.
33. Nanthaamornphong, A.; Carver, J.; Morris, K.; Filippone, S. Extracting UML Class Diagrams from Object-Oriented Fortran: Foruml. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Austin, TX, USA, 15-20 November 2015*; Hindawi Publishing Corporation, Scientific Programming: London, UK, 2015; pp. 1–15.
34. Anonymous. Dataset of the Paper "Automatically Classifying UML Class Diagrams from Images using Deep Learning". Zenodo. 2021. Available online: http://doi.org/10.5281/zenodo.4252890 (accessed on 4 May 2021).